An Interview with Michael Foord on IronPython

by Michael Foord

Blog: http://www.voidspace.org.uk/python/weblog/index.shtml
Book: http://www.manning.com/foord
Job: Senior Software Engineer and Community Champion
Employer: Resolver Systems http://www.resolversystems.com/
Publisher: http://www.manning.com/affiliate/idevaffiliate.php?id=282_94

The Python Papers> *How did you first get involved with software development?*

I've always loved programming. I started with Basic on the BBC microcomputer back when 32k was a lot of memory. I moved onto assembly language on the Amiga (a truly beautiful computer and operating system), and then took a break from programming for ten years.

I got involved with Python completely by accident about five years ago and immediately fell in love with the elegance and expressiveness of the language. I've been writing articles and involved in open source projects ever since.

TPP> *How did you get involved with IronPython?*

About two years ago I started working for Resolver Systems, a new startup in London. They were (well... are) developing a new spreadsheet application aimed at the financial services market.

Since this market is very conservative technologically, Resolver had to be built on an already accepted platform. This basically meant Java or .NET and for a desktop application .NET was the logical choice. The two developers assumed they would be working with C#, the default .NET language. Having an embedded interpreted language is a core part of Resolver and so they started to evaluate .NET scripting language engines. At the time IronPython was at version 0.7, but Giles and William were very impressed with it - particularly the level of integration with the .NET framework. They decided to see how far they could get writing Resolver in IronPython. I was the fourth Resolver developer and the first with any experience of Python!

Two years later, Resolver consists of 30 000 lines of IronPython code, plus another 100 000 lines of Python in the test framework.

Resolver is a great application for anyone who does data processing with Python or spreadsheets. Our public beta is now available and you can download it from our website.

TPP> *IronPython is just for Windows users, right?*

IronPython runs on both .NET and Mono. Mono is cross platform, running on various operating systems like Linux Mac OS X, Solaris, NetBSD, FreeBSD, OpenBSD and even Windows.

There is a community distribution of IronPython created by Seo Sanghyeon and called FePy (http://fepy.sourceforge.net). This includes patches for Mono compatibility and comes with the full Python standard library.

The Mono VM is an impressive platform, and there is a surprisingly healthy community of people who use IronPython on non-Windows platforms.

Advantages of IronPython over CPython, whichever VM you are running on, include:

- *Performance.* Some aspects of IronPython run much faster than CPython, particularly because of the just-in-time compiler.
- *Access to the .NET libraries, including third party extensions.* For Windows development the Windows Forms user interface library is better than anything I have seen available for Cpython. *[There are gtk and other interface libraries available for linux and also Windows - Ed]*
- *No global interpreter lock.* Multi-threaded programs can take advantage of multi-core processors, which they can't on Cpython.
- *Unicode strings.* This is coming in Python 3, but really does make working with text more pleasant.
- *.NET may be an easier corporate sell than Cpython*.
- *IronPython is dramatically easier to extend with C# than CPython is with C.*
- *The .NET platform supports a wide range of 'native' languages* (C#, VB.NET, and functional languages like F# and Nemerle just to name a few). Inter-operation between these languages is basically straightforward. The new DLR (Dynamic Language Runtime) extends this range of languages to include IronRuby, managed Javascript, Smalltalk and a port of Lua called Nua.
- *Through Silverlight the browser can be scripted with IronPython*.

> ***TPP>*** *You work for Resolver Systems, who are also featured in this issue. Could you describe how you have found the experience of software development using IronPython? How does it compare with other languages you have used professionally?*

My development experience is almost entirely with Python (plus a smattering of Javascript which is hard to avoid these days). The nice thing about developing with IronPython is that it is just Python, but you have this whole range of new libraries to use.

Working with IronPython was my first experience of using Windows APIs. I really didn't know what to expect and feared the worst! To my surprise the majority of the .NET APIs are very pleasant. A few are a bit over-engineered and make you jump through hoops, but in general they are very good to use - almost Pythonic (except for all the camel case method names and static methods instead of functions).

Since working at Resolver I have also regularly used C#. It is a fairly nice static typed language and is sometimes described as 'Java done right'. I have actually read a lot more C# than I have written. The .NET documentation, and most of the online examples, all use C# (which is why I created the IronPython Cookbook). It is very easy to read and not much harder to write, though I still prefer Python. Although they are very different languages, the core object models are similar which helps.

> ***TPP>*** *You have been writing a book called "IronPython in Action" for Manning Publications Co. How did you come to work with them?*

I first got in touch with Manning by reviewing a couple of books from their "In Action" series on my blog. I've always enjoyed writing, and after programming at Resolver for a few months I thought IronPython would be the good subject for a book. I approached them, and over six months or so we hammered out a proposal for "IronPython in Action". They've been very good to work with and I would recommend them to any aspiring technical writers.

> TPP> We have been lucky enough to be able to include a sample section in this

issue. Perhaps you'd just like to introduce it...

This is an excerpt from the first chapter. It is an introduction to IronPython for Python programmers and explains why Python programmers should be interested in IronPython.

TPP> *A lot of our readers will use Linux systems. IronPython can run through Mono. Will your book have something that Linux users will be able to make use of?*

Most of the book will be relevant to Mono. There is very little that is Windows specific in the book, except perhaps for one chapter!

One of the outcomes of the recent Mono summit was the Mono team showing off the native drivers for OS X, accessible through both Windows Forms and GTK#. This means that IronPython is a viable choice for writing genuinely cross platform applications. Windows Forms gets quite a lot of coverage in the early chapters of the book as it is used for the example application. Most of the rest of the examples, including embedding and extending IronPython, should 'just work' on Mono.

Seo Sanghyeon is helping review the book, so if there is anything that is different on Mono, or doesn't work, then hopefully he will point it out for me.

TPP> *What are the major communities for IronPython support and cooperation?*

Central to the community are obviously the IronPython and FePy websites. On the IronPython site you can report bugs and vote on ones that are important to you:

http://www.codeplex.com/IronPython
http://fepy.sourceforge.net

Most of the support happens on the IronPython mailing list. There is a very friendly community there with a good combination of those using IronPython on .NET and Mono. It's the best place to go for answers to specific questions:

http://lists.ironpython.com/listinfo.cgi/users-ironpython.com

Other useful places for information and articles are:

The IronPython Cookbook - http://www.ironpython.info/
A wiki with examples of using the .NET framework with IronPython, including embedding IronPython.

IronPython-Urls - http://ironpython-urls.blogspot.com/
A blog that covers the IronPython world linking to news and articles.

My pages on IronPython and Silverlight -
http://www.voidspace.org.uk/ironpython/index.shtml

TPP> *When is your book due to be published?*

I hope to finish writing the book early in the new year. It will be available in hardcopy two or three months after that.

In the meantime the first six chapters are already available via the Manning Early Access Program. Another four chapters will be added very soon. I'm particularly proud of the chapters on testing with IronPython, metaprogramming with IronPython and the integration with the .NET framework. There will also be a chapter on ASP.NET with IronPython written by my colleague Christian Muirhead. He has several years development experience writing web applications with both Python and .NET and is also writing the chapter on databases and web services.

Book Exerpt: “1.1.3 IronPython for Python Programmers”

We were lucky enough to gain permission to reproduce a sample section of Michael's upcoming book.

As I mentioned before, IronPython is a full implementation of Python 2.4. If you've already programmed with Python there is nothing to stop you experimenting with IronPython straight away.

The important question is; why would a Python programmer be interested in using IronPython? The answer is basically twofold, the platform and the platform. Let me try and make a bit more sense. First of all I mean the underlying platform that IronPython runs on; the CLR. Secondly, along with the runtime comes the whole .NET framework, a huge library of classes a bit like the Python standard library.

There are several reasons why the Common Language Runtime is an interesting platform. The CLR has had an enormous amount of work to make it fast and efficient. Multithreaded programs can take full advantage of multiple processors, something that CPython programs can't do because of a tricky creature called the 'GIL'[1]. Because of the close integration of IronPython with the CLR, extending IronPython through C# code is significantly easier than extending CPython with C. There is no C API to contend with, you can pass objects back and forth across the boundary without hassles and with no reference counting[2] to worry about. On top of all this, .NET has a concept called 'AppDomains'. These allow you to run code with reduced privileges, like preventing it from accessing the file system, which is a feature that has long been missing from CPython.

IronPython uses .NET classes natively and seamlessly, and there are a lot of them. Two of the gems in the collection are Windows Forms and the Windows Presentation Foundation, which are excellent libraries for building attractive and native looking user interfaces. As a Python programmer, you may be surprised by how straightforward the programmers interface to these libraries feels. Whatever programming task you are approaching, it is likely that there is some .NET assembly available to tackle it. This includes third party libraries for sophisticated GUI components, like data grids, where there is nothing comparable available for CPython. Table 1.1 shows a small selection of the libraries available to you in the .NET framework.

Table 1.1 Common .NET Assemblies and Namespaces

Assembly Name	Purpose
System	Contains the base .NET types, exceptions, garbage collection classes and much more.
System.Data	Classes for working with databases, both high and low level.
System.Drawing	Provides access to the GDI+ graphics system.
System.Management	Provides access to Windows management information and events (WMI), useful for system administration tasks.

System.Environment	Allows you to access and manipulate the current environment, like command line arguments and environment variables.
System.Diagnostics	Interact with processes.
System.XML	For processing XML, including SOAP, XSL/T and more.
System.Web	The ASP.NET web development framework.
System.IO	Contains classes for working with paths, files and directories. Includes classes to read and write to filesystems or data-streams, synchronously or asynchronously.
Microsoft.Win32	Classes that wrap Win32 common dialogs and components including the registry.
System.Threading	Classes needed for multithreaded application development.
System.Text	Classes for working with strings (like StringBuilder) and the Encoding classes which can convert text to and from bytes.
System.Windows.Forms	Provides a rich user interface for applications.
System.Windows	The base namespace for WPF, the new GUI framework that is part of .NET 3.0.
System.ServiceModel	Contains classes, enumerations, and interfaces to build Windows Communication Foundation (WCF) service and client applications.

As we go through the book we'll use several of the common .NET assemblies, including some of those new to the .NET 3.0 release. More importantly we'll learn how to understand the MSDN documentation so that you are equipped to use *any* assembly from IronPython. We will also do some client-side web programming with Silverlight, scripting the browser with Python. This is something that has not been possible before IronPython and Silverlight.

Most of the Python standard library works with IronPython; ensuring maximum compatibility

is something the Microsoft team has put a lot of work into. Do beware though. Not all of the standard library works; C extensions don't work because IronPython isn't written in C. n some cases, alternative wrappers may be available[3], but parts of the standard library and some common third party extensions don't work yet. If you are willing to swap out components with .NET equivalents, or do some detective work to uncover the problems, it is usually possible to port existing projects.

Where IronPython really shines is with new projects, particularly those that can leverage the power of the .NET platform. In order to take full advantage of IronPython there are a few particular features you will need to know about. These include things that past experience with Python alone won't have prepared you for. Before we turn to actually using IronPython, let's first look at how it fits in the world of the .NET framework.

---

[1] The 'Global Interpreter Lock', which makes some aspects of programming with Python easier but has this significant drawback.

[2] CPython uses reference counting for garbage collection, which extension programmers have to take account of.

[3] Several of these are provided by IPCE – the IronPython Community Edition. We'll look at this in a later chapter.